Tutorial on
VERIFICATION OF PROTOCOLS

by

Björn Pehrson

SICS R88012
Research Report
ISSN 0283-3638

Tutorial on

VERIFICATION OF PROTOCOLS

Björn Pehrson

Swedish Institute of Computer Science
Box 1263, S-164 28 Stockholm, Sweden
bjorn@sics.se

SICS Research Report R88012

Tutorial on VERIFICATION OF PROTOCOLS

Björn Pehrson
Swedish Institute of Computer Science
Box 1263, S-164 28 Stockholm, Sweden
bjorn@sics.se

ABSTRACT

The paper is a tutorial on some formal methods for verification of communication protocols. We focus on methods to show that a proposed communication protocol meets its specification by proving safety and liveness properties. Emphasis is put on finite state spaces for which algorithms are available which can be used to mechanise property proving.

1. INTRODUCTION

Validation of communication protocols has several aspects. Three important aspects are illustrated in figure 1 (arrows directed upwards).

The objective might be to investigate if a specified service is what the user really had in mind. This *validation* task can often be solved by simulation or testing using a rapid prototype of the service definition or the protocol and by having the user to pass his verdict. Since human judgement is involved, formal methods are not easily applicable.

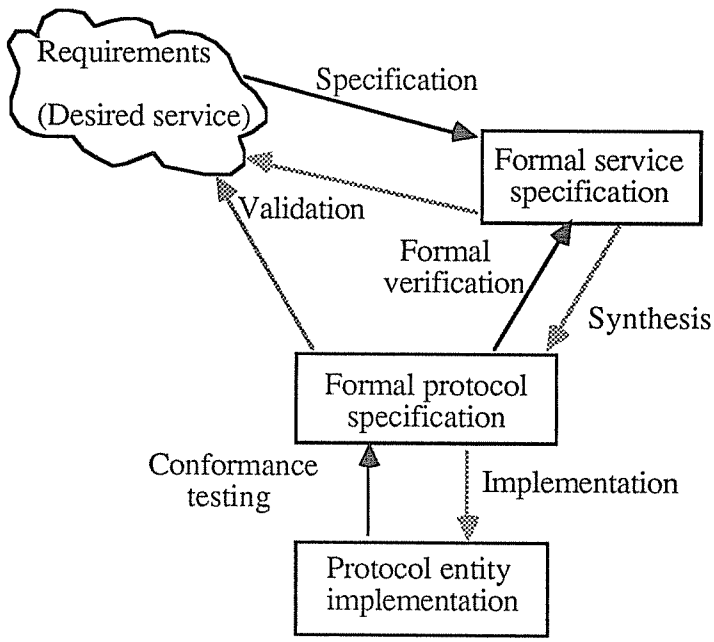


Fig.1 Different aspects of protocol validation

Another objective might be *verification* of a proposed protocol against a service specification, i.e. to prove that a proposed *protocol* provides a specified *service*, e.g. that the protocol possesses desirable and lacks undesirable properties (property proving). Using formal models, this task can sometimes be attacked by formal reasoning. This objective is important to meet in the standardization of new communication services and protocols.

Yet another objective might be to test that an alleged implementation of a *protocol entity* actually conforms to the protocol specification. This kind of *conformance testing* has received a lot of attention lately due to the political promotion of software product certification schemes.

In this paper, we will deal only with the second aspect, verification of protocol specifications based on formal reasoning. We will emphasise methods for which it is possible to find algorithms so that programs can be written that automate the property proving. We will not, however, discuss tools. For some methods, tools are available and several projects are under way. A survey of tools and tool development projects can be found in [Bochmann87]. We will restrict our discussion to time-independent functionality and only give some references to verification of performance aspects. An earlier introduction to protocol verification can be found in [Bochmann80].

Protocols and their properties

We will follow the OSI-model [ISO 7498] in which the service in each layer is provided by means of a protocol described as the procedure for communication between two protocol entities using the service of the underlying layer (fig. 2)

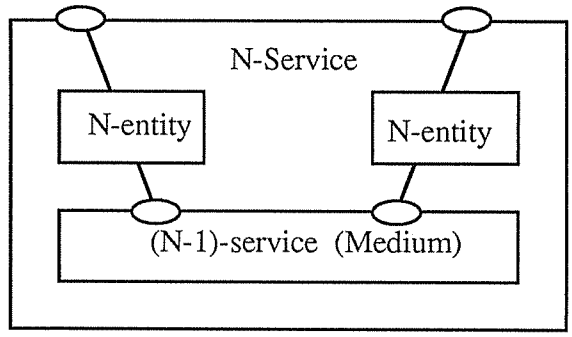


Fig.2 The structure of an OSI-layer

The trend in the development of the state of the art in protocol verification has been an increasing degree of formalisation. The essence of a protocol is the behaviour of the communicating entities and the underlying medium, i.e. the set of possible sequences of events that can occur when the entities execute. There are different approaches to specification of protocols and their properties. Often behaviours are specified as some kind of transition system, eg state machines, nets, traces, etc.

Protocol properties are often specified in terms of assertions associated with a transition system. Protocol properties are classifed in the categories **safety** properties and **liveness** properties. The emphasis is put on properties related to the ordering of events. Safety properties are properties that assure that no undesirable events will ever happen (e.g. undefined receptions, deadlocks). Safety properties specify only what may or may not happen but do not require that anything ever does happen. Liveness properties assure that desired events will eventually happen (e.g. completion, recurrence, progress). Safety properties can be stated as invariant assertions associated with transition system models or as relations (simulation, bisimulation) between a proposed and a required behaviour. Liveness properties, in general, require temporal reasoning, e.g. in temporal logic.

Exposition

Since this is a tutorial, and not a survey, we have no intention to cover the state of the art completely. We have chosen our examples entirely from a pedagogical point of view, without special regard to completeness or to cover the milestones. Our goal has been to discuss some principal protocol verification methods which have proved to be useful in practical applications.

Formal reasoning about a protocol requires some sort of formal description of the protocol and its properties. The description should also include the underlying service, i.e. the (N-1)-service, and the service to provide, i.e. the N-Service. In section 2, we will introduce the basic notations used in the sequel. We will also describe two sample protocols which will be used later to illustrate some of the verification methods, the alternating bit protocol and the sliding window protocol. The use of these protocols for demonstration has sometimes been criticised since they are too simple to illustrate the potential of the methods in applications of realistic complexity. However, since it is easier to understand basic principles from simple examples, we will discuss how the methods scale up separately .

Different description techniques provide different support for reasoning. It is therefore natural to discuss verification methods according to the underlying formal description technique. Systems having a finite state space take a special position. The verification methods can be based on an exhaustive exploration of the state space and it is often possible to find algorithms which make it possible to mechanically determine properties. In the general case, when the state space is infinite, this is not possible since most problems are undecidable. Other proof methods must be used, e.g. induction. The proof must be designed by hand and a certain ingenuity often is required to find the proof.

In section 3, we will review classical reachability analysis of communicating finite state machines. This verification method has attracted much attention during the years and is probably still the most widely used. Many applications have been reported. Different refinements have been made to reduce the problem of state explosion, some of which will be discussed.

We also discuss some algebraic theories, sometimes referred to as process algebras. They have proved to be useful, both as formalization of the classical methods and as bases for a broader scope of properties to be analyzed. One of these theories, CCS, has played an important role in the development within ISO of LOTOS, a formal description language for OSI-protocols. Several applications to realistic protocols have been reported.

In the methods discussed so far, liveness properties cannot readily be expressed. An approach to this problem is to add liveness conditions expressed as temporal logic formulas and check if the state transition system is a model for these formulas. The method is called model checking. It will be discussed in section 4.

In section 5, we will discuss some more general methods which deal with infinite state spaces. The general concurrent program verification problem is considerably more complex than the finite state case. It is beyond the scope of this tutorial to go into detail on these general verification methods. We will only give a short introduction and give pointers for further study.

2. FINITE STATE TRANSITION SYSTEMS

In this section, we will define the basic notation and sample protocols to be used in the following examples.

Notation

We will later use simple data link protocols to illustrate some of the verification methods. We refine figure 2 by dividing the medium into two uni-directional channels (figure 3). A Sender entity transmits messages to a Receiver entity via one channel. Acknowledgements are transmitted back by the Receiver to the Sender on the other channel.

Each module represents a *process*. A process can have an associated *behaviour* and/or consist of a *parallel composition* of component processes. In figure 3, the processes are: the service provided (the outer box), the protocol entities, and the medium channels which describe the underlying service. The service provided is also a parallel composition of the others.

Behaviour
We define the behaviour of a process in terms of a simple *transition system*, a communicating finite state machine (cfsm). A cfsm is a tuple <S, i, E, T> where S is a set of states, i∈ S is the initial state in which the system starts. E is a set of events and includes *communication events* (input and output) and *internal events*. T is a set of labeled transitions (in SxExS). The labels are events in E. When an event in E occurs, a transition in T takes the system from one state in S to another. All events are atomic. The communication events are the only externally visible and are used for synchronization of different processes. Communication events can only occur in coevents as defined below. In the graphical representation states are represented by circles, transitions as directed arcs (arrows) and the initial state as a black token.

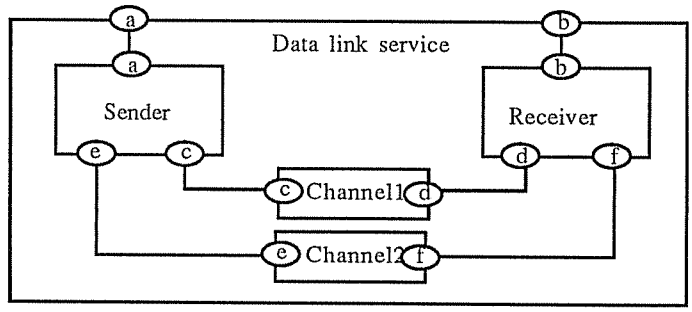


Fig.3 A data link layer

Parallel composition
A parallel composition of processes means that the processes run in parallel. There are two different semantic models for composition of parallel event sequences: *interleaving* semantics, where the events in the parallel sequences are totally ordered in all possible interleavings, and *true parallelism*, where the events are only partially ordered by synchronizing coevents.

Each process has a finite number of distinctly named ports at which communication events can occur. The ports can be used to interconnect processes in a parallel composition by giving two different ports the same name. Two different ways to communicate via ports will be discussed, *synchronous* and *asynchronous*.

Message transmission between ports is *synchronous* if the transmitting process cannot transmit anything until the receiving process is ready to accept it as input. This is also referred to as *direct coupling* [Bochmann77]. The synchronization is described in terms of *coevents*. Here, we will define a coevent as a tuple of communication events including one output event and at least one input event, e.g. <p!c, p?d>, which form an atomic action, where p is a port name, ! can be interpreted as an output event, ? as an input event, and c and d are values. If the values are equal, the coevent is enabled. If the coevent is enabled and occurs, both associated transitions occur simultaneously and the common value is considered to be passed. This is a simplified value passing mechanism which is sufficient for our purpose. More general mechanisms can be defined. If more than one input port can synchronize directly with an output event in a coevent, we talk about *multicasting* or *broadcasting*.

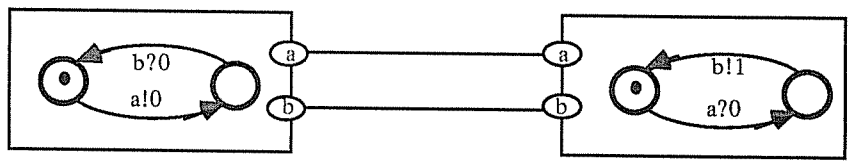


Fig.4 Synchronously communicating processes

The processes in figure 4 can only perform one transition fired by the coevent <a!0,a?0>. The values involved in the next communication events are not equal.

Message transmission between ports is *asynchronous* if the transmitting process can transmit as soon as it is ready to do so,without having to wait for the receiving process. Asynchronous communication requires an unbounded buffer if no messages are to be lost.

Compositionality, Global states and Abstraction

We say that our description technique is *compositional* if we can determine the behaviour of the parallel composition from the behaviours of the components. In the notation above this means to construct the cfsm of the composition from the component cfsm's. This is done by forming a cfsm on the *global* state space S (sometimes called reachability graph), a subset of the the Cartesian product of the component state spaces. Global states, including the initial state, become tuples of component states. To form E and T, all possible transitions are explored. All events in each coevent, and associated transitions, are reduced to an internal (silent) event associated to a single transition in the global state space. The internal transitions are not observable externally in terms of communication events.

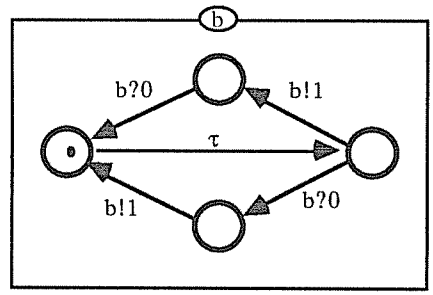


Fig.5 The global state space resulting from a parallel composition of the cfsm's in fig 4

In figure 5, the event τ denotes the silent (internal) event resulting from the coevent <a!0, a?0> which is the first event that can occur in figure 4. If we assume that only two events can participate in a coevent, the a port becomes externally unvisible.

Sample protocols

To illustrate the verification methods, we will use some data link protocols. The protocols are used to provide reliable data link services over media which consist of unreliable transmission lines, i.e. lines that can cause transmission errors, such as lost or garbled messages.

From a user point of view, a data link or a transmission line could be regarded as queue. Messages are enqueued in one end and can, after some delay, become dequeued in the other end. If there is only one place in the queue, a simple service specification could look like in figure 6.

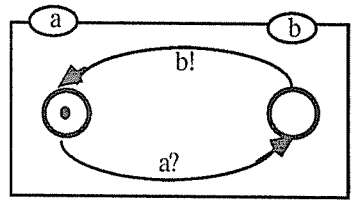


Fig.6 Data link service defined as a state machine

Some protocols allow more than one message to be in transit in the medium simultaneously. To model a queue with n places, we need n+1 states. The example in figure 7 describes a queue with 3 places. The internal event firing a loss transition from state 1 back to the initial state describes that a message may be lost when there is only one message in the queue.

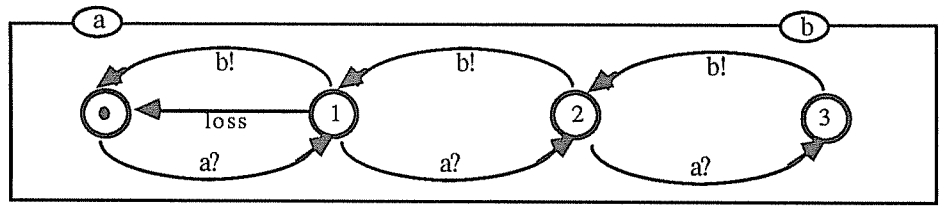


Fig.7 Description of an unreliable queue

Some protocols may even allow an unbounded number of messages to be in transit. This could not be handled using finite state decriptions. Sometimes it is unimportant how many messages are in transit in the medium, or, as a first analysis of the protocol, we might want to simplify our problem. To do this, we could short-circuit the medium and identify its input and output ports. However, we might still want to include medium properties like unreliability, e.g. indeterministic loss of messages. One way to include such properties is to include them in the description of the protocol entities. We could also reduce the medium to just one state and describe the reception and delivery of a message as an atomic action as in figure 8. The coevent concept defined above must then be extended to include all involved communication events.

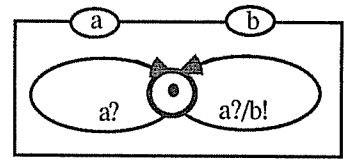


Fig.8 The empty medium abstraction

The notation a?/b! means an atomic sequential composition of the communication events a? and b!. This atomic sequential composition is used here to simplify our example. It should be used carefully, since it may exclude vital transition sequences and cause deadlock. The left transition, labeled a?, can fire as soon as there is an a! enabled in the environment. Since the input event is not followed by an output event, this transition represents loss of the message in the medium. Both transitions can fire when both a! and b? are enabled in the environment. The choice between them is indeterministic.

Data link protocols
In order to cope with an unreliable medium, the receiver has to acknowledge the receipt of correct messages. Garbled messages are detected by using some encoding technique, e.g. CRC. Messages which cannot be corrected are discarded and treated as lost messages.

To detect lost messages, each message can be numbered by associating a sequence number to the message. Periodically, until receiving an acknowledgement containing the same sequence number, the sender retransmits the message triggered by a time-out mechanism. Since also acknowledgements may be lost, the receiver has to retransmit acknowledgements until a message with the "next expected sequence number" arrives. Then, the receiver starts to acknowledge this new sequence number, and increments "next expected sequence number".

The alternating bit protocol
If the sender requires every message to be acknowledged before the next message is transmitted, only two values are needed for the sequence number, one for the latest acknowledged message and one for the latest transmitted message. One bit, alternating between 0 and 1, is consequently sufficient to represent the sequence number. This fact has given the protocol its name [Bartlett69]. The description in figure 9 is similar but not identical to one of the examples in [Bochmann78]. When the sender enters the states labeled s, a message has been received from the a port and is ready to be transmitted together with the associated sequence number. When the receiver enters the states labeled r, a message has been received from the sender and the transmission of an acknowledgement is pending.

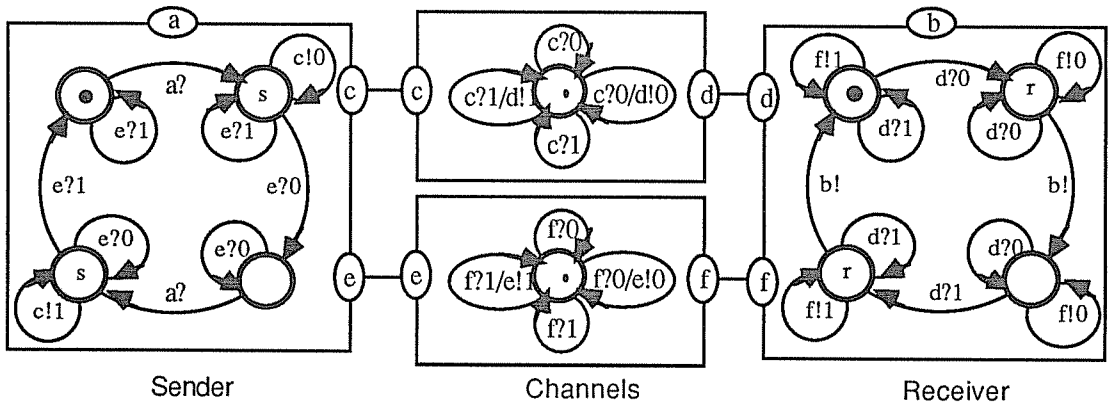


Fig.9 The alternating bit protocol over an empty medium abstraction

The sliding window protocol
If use the same mechanisms as above, but the sender accepts more than one message to be unacknowledged when transmitting a new message, the sequence number must have a range including all the outstanding messages and the latest previously acknowledged message. The sliding window protocol has been used by several authors to illustrate various verification methods in a more complex example, e.g. in [Bochman77, Jonsson87, Shankar83].

3. REACHABILITY ANALYSIS

The pioneering work in the field of protocol verification was based on reachability analysis of directly coupled finite state machines [Bochmann78], [Zafiropulo80]. The method has been refined and developed during the years and is probably still the most used method for protocol verification.

Reachability graph
Reachability analysis means exhaustive exploration of all possible interactions between a set of communicating processes modeled as state transition systems. It is a simple and straight forward method which is easily mechanized.The global state space, also called the reachability graph, of the set of communicating state machines is calculated and analysed.

In figure 10, the reachability graph of the alternating bit protocol specification in figure 9 is shown. The unlabeled looping transitions in each state represent loss and retransmission of messages and acknowledgements. The cd and fe events are internal events resulting from the coevents causing transfer of messages and acknowledgements. The involved sequence numbers are also indicated.

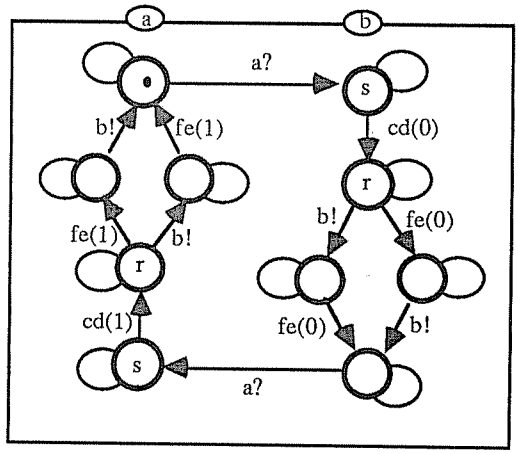


Fig.10 Reachability graph formed from figure 9

The classical method deals with safety properties only. Sometimes, the *possibility* of desired events to happen is also referred to as a liveness property. The more orthodox standpoint is that a liveness property is present only if it can be proved that the desired events actually*will* happen. Liveness properties in the orthodox sense cannot be expressed or verified using the methods discussed in this section. Examples of safety properties which could be verified are absence of

- undefined receptions
- unreachable or isolated states
- deadlocks, ie global states from which no escape is possible
- undesired loops, i.e. infinite loops in which no progress is made

Also, we could compare the behaviour of the composition to the service specification. It is obvious that the machine in figure 10 is different from the machine in figure 6. Yet, both figures describe an aspect of the same protocol. If only externally visible transitions are considered, i.e. a? and b!, the same event sequences are possible. We will discuss equivalence between state machines further in the section on process algebras below.

State space explosion
Evidently, the global state space can be calculated only if it is finite. Also, a problem with reachability analysis is the exponential complexity which leads to state space explosion. There are various relief techniques developed to deal with this problem. Some of these techniques are discussed in [Lin87], e.g. projection [Lam84], transition choice rules [Blumer86], different progress criteria [Gouda84, Gouda85].

A discussion about the limits of reachability analysis can be found in [Holzmann87]. He claims that his carefully designed validator handles protocol descriptions that generate 10^5-10^7 states in the order of minutes of CPU-time on a smaller machine (Vax 11/750) and seconds of CPU-time on a larger machine (Cray), and that this most likely covers a large fraction of practically interesting protocols.

Examples of protocols partially verified by traditional reachability analysis include X.21 [West78] and X.25 [Bochmann78].

Process algebra
The term process algebra is sometimes used to denote a number of algebraic theories dealing with communicating processes. One of these theories which has been widely used to specify and verify protocols is CCS, a Calculus of Communicating Systems [Milner80]. CCS can be regarded as a formalization of the traditional finite state machine approach into an algebraic framework. A tutorial on CCS is available in [Walker87].

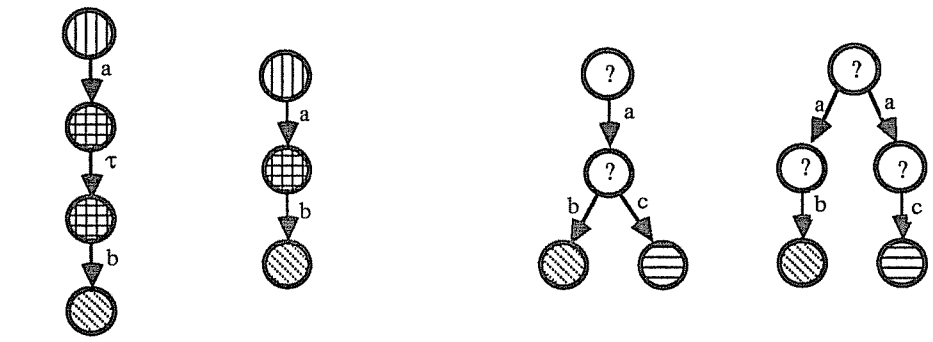
In CCS, processes are called agents. The semantics of CCS is defined in terms of transition systems, e.g. trees or charts. The original CCS syntax is a textual linear language in which expressions are formed. Agents are defined from communication events, internal events, and a set of operators like action prefix, which is a limited form of sequential composition (.), parallel composition (|) , and non-deterministic choice (+). Recursion is used to define cyclic behaviours.

Parallel composition of agents is similar to communicating state machines as discussed above. The restriction operator (\) is used to hide ports which are not to be available for external interaction. The expansion theorem in CCS corresponds to forming the global state machine. The theorem provides an algorithm for this purpose. In the expansion, coevents form a silent event, τ. Expansion and restriction provide means to abstract from internal events.

Observation equivalence between two agents is proven by the existence of a *bisimulation* relation [Milner83] between the state sets of the agents. A bisimulation R has the following property: whenever p$\boldsymbol{R}$s (the states p and s are related by $\boldsymbol{R}$) and p $\xrightarrow{a}$ p' (p can reach p'

through a sequence of transitions with the observable content a), then s $\xrightarrow{a}$ s' with p'$\boldsymbol{R}$s', and symmetrically if s $\xrightarrow{a}$ s' then p $\xrightarrow{a}$ p' with p'$\boldsymbol{R}$s'. Intuitively, this means that each transition of p can be simulated by a transition of s and vice versa. Hence the states p and s can never be distinguished by an external observer.

In figure 11, we illustrate the bisimulation concept. The agents in 11a), where τ is an internal event, are observational equivalent, the agents in 11b) are not. States which bisimulate eachother are marked similarly.



a) observational equivalent agents b) not equivalent agents

Fig. 11 Illustration of bisimulation

The alternating bit protocol
The data link service in figure 6 will become as follows when specified in CCS:

Service = a?.b!.Service

To simplify the specification of the alternating bit protocol in figure 9, we use our own message passing convention as described above rather than the one defined in [Milner80]. As atomic sequential composition is not available in CCS, we do not use the empty medium abstraction but consider media of capacity one element.

Sender = send0
send0 = a?.rep0 + e?1.send0
rep0 = e?0.send1 + c!0.rep0 + e?1.rep0
send1 = a?.rep1 + e?0.send1
rep1 = e?1.send0 + c!1.rep1 + e?0.rep1

Receiver = rec0
rec0 = d?0.ack0 + f!1.rec0 + d?1.rec0
ack0 = b!.rec1 + f!0.ack0 + d?0.ack0
rec1 = d?1.ack1 + f!0.rec1 + d?0.rec1
ack1 = b!.rec0 + f!1.ack1 + d?1.ack1

Channel1= c?0.d!0.Channel1 + c?1.d!1.Channel1 + c?0.Channel1 + c?1.Channel1

Channel2= f?0.e!0.Channel2 + f?1.e!1.Channel2 + f?0.Channel2 + f?1.Channel2

We form the parallel composition of the agents

Protocol = (Sender | Channel1 | Channel2 | Receiver)\{c,d,e,f}

The expression is expanded using the expansion theorem. Here, we will have the same

problem with state explosion as in the cfsm case. Finally, we decide if the result is observational equivalent to the predefined service agent:

An overview of methods to decide observational equivalence can be found in [Bolognesi87]. An illustrative application of CCS to verification of the CSMA/CD medium access protocol is reported in [Parrow88].

Net models
There are many variations on the theme of reachability analysis. A variant of the finite state machine approach uses Petri nets and different extensions. Good surveys and discussions of Petri Net based methods for protocol verification can be found in [Diaz82] and [Courtiat84].

4. MODEL CHECKING IN TEMPORAL LOGIC

The model checking approach to verification of finite state concurrent systems [Lichtenstein85], [Clarke86], [Vardi86] can be regarded as an extension of the methods discussed in section 3. Propositional temporal logic formulas are used to specify desirable protocol properties associated with the states in the global state transition system describing the protocol. This provides a more abstract way to specify and reason about properties than do pure transition systems. Also, the method provides means to specify and reason about liveness properties in the orthodox sense.

Temporal logic is a modal logic, i.e. a logic which distinguishes among different states by using modal operators. It is possible to speak about a proposition being true now, eventually or henceforth, from the next state and henceforth, etc. In branching time temporal logic, the semantic model is tree-oriented. The quantifiers $\exists, \forall$ are used to indicate branches in the tree. Linear time logic has a trace oriented semantics.

CTL
Clarke, Emerson and Sistla [Clarke86] use a propositional branching time temporal logic called *computation tree logic* (CTL).

The syntax of CTL is defined on a set of atomic propositions. Every atomic proposition is a CTL formula. If f_1 and f_2 are CTL formulas, so are $\neg f_1$, $f_1 \wedge f_2$, AXf_1, EXf_1, $A[f_1 \, U \, f_2]$, and $E[f_1 \, U \, f_2]$, where $\neg$ and $\wedge$ have their usual meaning. X is the *next* operator. A is short for *always* and E for *eventually*. AXf_1 (EXf_1) intuitively means that f_1 holds in every (some) immediate successor of the current state. U is the *until* operator and $A[f_1 \, U \, f_2]$, ($E[f_1 \, U \, f_2]$) intuitively means that for every (some) computation path, there exists an initial prefix of the path such that f_2 holds at the last state of the prefix and f_1 holds at all other states along the prefix.

The semantics of CTL formulas is defined via a *structure* relating temporal logic propositions to a labeled state transition graph. A temporal logic structure is a tuple $M = (S, s_o, R, P)$, where S is a set of states, $s_o \in S$ is an initial state, R is a binary relation on S describing possible transitions, and P is an assignment of true atomic propositions to each state. A *path* is an infinite sequence of states where each transition belongs to R. To each structure M, there is an infinite *computation tree* with s_o as root and the arcs representing transitions in R. In an extended version of CTL, *fairness predicates* are used to select only fair paths. The fairness predicates express conditions that must hold infinitely often.

Truth in a structure is indicated with the *validity relation* $\models$. The relation is defined inductively starting with the atomic propositions followed by operators of increasing complexity. This definition is also used in the design of the model checking algorithm which is discussed below.

$s_o \models p$ iff $p \in P(s_o)$

$s_o \models \neg f$ iff not$(s_o \models f)$

$s_o \models f_1 \wedge f_2$ iff $s_o \models f_1$ and $s_o \models f_2$

$s_o \models \boldsymbol{A}Xf_1$ iff for *all* states t such that $(s_o, t) \in R$, $t \models f_1$

$s_o \models \boldsymbol{E}Xf_1$ iff for *some* state t such that $(s_o, t) \in R$, $t \models f_1$

$s_o \models \boldsymbol{A}[f_1 \, \boldsymbol{U} \, f_2]$ iff for *all* paths $(s_o, s_1, ...)$, $\exists i[i \geq 0 \wedge s_i \models f_2 \wedge \forall j[0 \leq j < i \rightarrow s_j \models f_1]]$

$s_o \models \boldsymbol{E}[f_1 \, \boldsymbol{U} \, f_2]$ iff for *some* path $(s_o, s_1, ...)$, $\exists i[i \geq 0 \wedge s_i \models f_2 \wedge \forall j[0 \leq j < i \rightarrow s_j \models f_1]]$

The following derived operators are used in CTL:

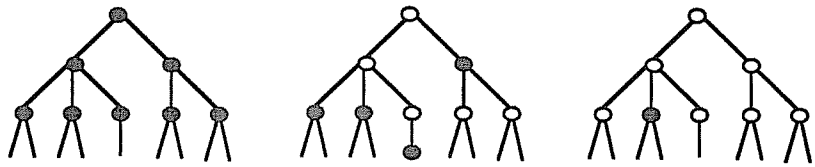
$\boldsymbol{AF}(f) \equiv \boldsymbol{A}[\text{true} \, \boldsymbol{U} \, f]$ intuitively means that f holds in the future along every path, i.e. f is *inevitable*

$\boldsymbol{EF}(f) \equiv \boldsymbol{E}[\text{true} \, \boldsymbol{U} \, f]$ means that there is some path leading to a state in which f holds, f is *potentially* true

$\boldsymbol{EG}(f) \equiv \neg \boldsymbol{AF}(\neg f)$ means that there is some path on which f holds

$\boldsymbol{AG}(f) \equiv \neg \boldsymbol{EF}(\neg f)$ means that f holds at every state on every path, i.e. f holds *globally*

The semantics of some of these operators is illustrated in figure 12 in terms of computation trees.



a. p global (**AG**p) b. p inevitable (**AF**p) c. p potential (**EF**p)

Fig. 12 Representation of some simple correctness properties in CTL (• =p, o = ¬p)

Algorithm
To determine if a formula f_o is true in the structure M, it has to be evaluated in each state. Each state is labeled with the set of subformulas of f_o that are true in the state, starting in stage 1 with subformulas of length 1, then stage 2 with subformulas of length 2, until stage n = length(f). Then, for all states s, M, $s \models f$ iff f is a label of s for all subformulas of f of f_o.
A discussion about methods to reduce the problem of state space explosion when forming the global state space can be found in [Clarke87].

The alternating bit protocol in CTL
The structure used in [Clarke86] to specify the alternating bit protocol consists of a state transition system, specified as a CSP program, and two atomic propositions, s and r. It is similar to our example in the figures 9 and 10. The state transition system corresponds to the cfsms, and the two atomic propositions corresponds to the labels s (a message has been received from the data link user and is ready to be transmitted to the receiver) and r (a message has been received from the sender and the transmission of an acknowledgement is pending).

We want to prove that sending a message strictly alternates with receiving a message and that the same message is received as was sent. To prove that s and r alternates is straight forward. To prove that the same message is received as was sent, we have to involve the messages

submitted, since the alternating bit mechanism is supposed to be hidden from the user. As CTL is a propositional logic (allowing only atomic propositions, no quantifiers over variables), we have to repeat the proof for each possible message. For simplicity, it is assumed that the message also consists of just one bit and can take the values 0 and 1.

The following CTL formulas imply that sending a message (s) strictly alternates with receiving a message (r) and that the same message is received (m_r) as was sent (m_s).

$$AG(r \rightarrow A[r\ U\ (\neg r \wedge A[\neg r\ U\ s])])$$

$$AG(s \wedge m_s{=}0 \rightarrow A[s\ U\ (\neg s \wedge A[\neg s\ U\ r \wedge m_r{=}0])])$$

$$AG(s \wedge m_s{=}1 \rightarrow A[s\ U\ (\neg s \wedge A[\neg s\ U\ r \wedge m_r{=}1])])$$

If only fair paths are considered, i.e. paths in which a message cannot be lost or garbled each time transmitted, the conjunction of the three formulas can be shown to be always globally true.The fairness constraints are formulated as a set of states, in this case the states labeled s and r. Any infinite path has to pass these states infinitely many times.

It is not trivial to get an intuitive feeling about the meaning of the formulas A discussion about the expressiveness of temporal logic can be found in [Schwartz82].

Other similar methods

Richier et al [Richier87] use a method in the Xesar system which is very similar to CTL. Vardi and Wolper [Vardi86] discuss the model checking method using a linear time temporal logic. It is applied to reasoning about probabilistic concurrent programs.

Parrow [Parrow86] takes a slightly different approach. Rather than checking a transition system against a temporal logic formula, temporal logic is integrated into the transition system calculus (CCS) to specify and reason about properties which are not easily expressed otherwise. The method is used to verify safety and fairness properties of the medium access protocol CSMA/CD.

5. INFINITE STATE SPACES

The general concurrent program verification problem is considerably more complex than the finite state case. Here, we will give a short introduction and give pointers for further study. A good exposition of basic concepts can be found in [Hailpern82].

Floyd's approach to reasoning about sequential programs [Floyd67] was formulated by Hoare [Hoare69] in an axiomatic framework, which was extended by Manna and Pnueli [Manna74] to deal with termination. Owicki and Gries developed the axiomatic approach of Hoare to include concurrent programs [Owicki76]. Other methods to reason about safety properties of concurrent programs were developed, e.g. by [Keller76, Lamport77]. Later, temporal logic was combined with some of these frameworks to deal with liveness properties, e.g in [Hailpern83, Manna81, Owicki82].

The ingredients, when verifying parallel programs in these methods, are inference rules for the constructs of the programming language, invariants expressing the safety properties of the program, e.g in terms of logic formulas that must be satisfied always, and liveness assertions. Several frameworks that are based on the same fundamental principles and integrates transition systems, classical logic and temporal logic more intimately have been developed later, e.g by [Lamport83, Nguyen86, Jonsson87, Reed88].

In special cases, it is possible to find proof methods, and even to prove decidability of some properties also in the infinite state space case. One such special case is if we allow an infinite number of messages in our channels of the alternating bit protocol in figure 9. Larsen and

Milner [Larsen86] use this example to illustrate a compositional proof method for communicating systems. Due to symmetry and regularity, it is possible to reduce the number of different channel states to be investigated to a finite number. The same example is used by Pachl [Pachl87], who introduces a formalism for reasoning about the contents of the communication channels, called channel expressions. The underlying theory is related to reachability analysis and can be used to prove a decidability result about deadlock detection in communicating finite state machines. He assumes, however, that any algorithm is too slow to be of interest in practice.

6. CONCLUSIONS AND TRENDS

We have discussed some different approaches to protocol verification. The formal description techniques used are state transition systems and assertional techniques based on some logic.

In the case when a protocol can be described with a finite state space, algorithmic verification of safety and liveness of behaviour is possible. According to Holzmann [Holzmann87] most realistic applications fall into this category. A number of successful applications of realistic size has been reported. Tools supporting such algorithmic methods are being developed [Bochmann87]. Careful structuring of the specification to be verified is vital to decrease the complexity and facilitate verification. This conclusion is made from a theoretical analysis, e.g in [Shankar83] where changes in the HDLC protocol are proposed. The experience from a large industrial protocol standardization project confirms this view [Hansson88].

In the case of an infinite state space, the methods require more intimate knowledge both about proof methods, and about the design to be verified. Proofs are not always possible to find. Further research is needed to make these methods more widely available.

Also, there are other interesting protocol properties to verify, which we have not discussed in this tutorial, e.g. related to time and performance. Several authors have presented different approaches to specification and reasoning about time and performance, e.g. [Rudin83, Shankar83, Zic87, Masapati87].

A new trend in reasoning about protocols is to use concepts and methods introduced in the area of knowledge based systems. Some early work in this area can be found, e.g. in [Neiger87, Juanole88].

We have discussed verification without questioning its role in the design process. To have something to verify, we must come up with a design. Of course, it would be better to develop provably correct synthesis procedures which could be used to derive protocol specifications from specifications of the service to provide and the service available. However, the synthesis problem is much more difficult, and we have to use all methods available to increase our confidence in a specification before implementation. Time and money is saved in design and maintenance, and the final product becomes more reliable.

ACKNOWLEDGEMENTS

I want to thank Bengt Jonsson and Joachim Parrow for enlightning discussions as well as Lars Kahn and Fredrik Orava for proof reading and giving comments on the manuscript.

REFERENCES

[Bartlett69] K.A.Bartlett, R.A. Scantlebury, P.T. Wilkinson, A Note on Reliable full-duplex Transmission over half-duplex Links, CACM 12,5 (1969), 260-261

[Blumer86] T.P Blumer, D.P. Sidhu, Mechanical Verification and Automatic Implementation of Protocols, Proc. ACM SIGCOMM, 1983

[Bochmann77] G.V.Bochmann, R.J.Chung, A Formalized Specification of HDLC Classes of Procedures, Proceedings of the National Telecommunication Coference, Los Angeles, California, December 1977

[Bochmann78] G.V.Bochmann, Finite State Description of Communication Protocols, Computer Networks, Vol.2, October 1978, pp361-372.

[Bochmann80] G.V.Bochmann, C. Sunshine, Formal methods in Communication Protocol Design, IEEE Trans. on Comm., Vol.COM-28, No.4, April 1980

[Bochmann87] G.V. Bochmann, Usage of Protocol Development Tools: The Results of A Survey, IFIP Int. Conf. on Protocol Specification, Testing and Verification VII, Zürich 1987, North Holland 1988

[Bolognesi87] Tommaso Bolognesi, Scott A. Smolka, Fundamental Results for the Verification of Observational Equivalence - a Survey, IFIP Int. Conf. on Protocol Specification, Testing and Verification VII, Zürich 1987, North Holland 1988

[Clarke86] E.M.Clarke, E.A.Emerson, A.P.Sistla, Automatic Verification of Finite-State Concurrent Systems using Temporal Logic Specifications, ACM TOPLAS Vol.8, #2, April 1986

[Clarke87] E.M.Clarke, O Grümberg, Avoiding the State Explosion Problem in Temporal Logic Model Checking Algorithms, Proc. 6th Annual ACM Symp. on Principles of Distributed Systems, Vancouver, August 1987.

[Courtiat84] J.P.Courtiat, J.M.Ayache, B.Algayres, Petri Nets are Good for Protocols, Proc. ACM SIGCOMM, June 1984

[Diaz82] Michel Diaz, Modeling and Analysis of Communication and Cooperation Protocols using Petri Net Based Models, Proc. IFIP Int. Conf. on Protocol Specification, Testing and Verification II, Idlewyld 1982, North Holland 1982

[Floyd67] Robert W. Floyd, Assigning Meaning to Programs, Proc. Symp. in Applied Math XIX, pp 19-32, American Math. Society, 1967

[Gouda84] M.G.Gouda, Y.T. Yu Protocol Validation by Maximal Progress State Exploration, IEEE Trans. Comm, COM-32(1):94-97, January 1984

[Gouda85] M.G.Gouda, J.-Y. Han, Protocol Validation by Fair Progress State Exploration, Computer Networks and ISDN Systems 9:353-361, 1985

[Gouda86] M.G.Gouda, C.-K.Chang, Proving Liveness for Networks of State Machines, ACM TOPLAS, vo.8,#1, January 1986

[Hailpern82] Brent T. Hailpern, Verifying Concurrent Processes Using Temporal Logic, Springer LNCS 129

[Hailpern83] Brent T. Hailpern, Susan S. Owicki, Modular Verification of Computer Communication Protocols, IEEE Trans. on Comm., Vol.COM-31, No.1, January 1983

[Hansson88] H.Hansson, F.Orava, B.Pehrson, Specification and Validation of Services and Protocols for a Public Land Mobile ISDN System, IEEE/EUREL 8th European Conf. on Electrotechnics, Stockholm 1988

[Hoare69] C.A.R. Hoare, An axiomatic Basis for Computer Programming, CACM, 12(10), October 1969.

[Holzmann87] Gerard J. Holzmann, On Limits and Possibilities of Automated Protocol Analysis, IFIP Int. Conf. on Protocol Specification, Testing and Verification VII, Zürich 1987, North Holland 1988

[ISO 7498] The Reference model for Open System Interconnection, ISO International Standard IS7498.

[Jonsson87] Bengt Jonsson, Modular Verification of Asynchronous Networks, Proc. 6th Annual ACM Symp. on Principles of Distributed Systems, Vancouver, August 1987.

[Juanole88] Guy Juanole, Omar Amyay, Pierre Azema, Rune Gustavsson, Björn Pehrson Towards a Knowledge Base for Design of (N)Service/Protocol Pairs - An Epistemic Approach, EUTECO 88, North Holland 1988.

[Lam84] Simon S. Lam, A. Udaya Shankar, Protocol Verification via projections, IEEE Trans. on Comm. SE-10(4):325-342, July 1984

[Lamport83] Leslie Lamport, Specifying Concurrent Program Modules, ACM TOPLAS 5, 2 (1983), pp 190 - 222

[Lamport83] Leslie Lamport, A Simple Approach to Specifying Concurrent Systems, SRC Report #15, Digital Systems Research Center, 1986

[Larsen86] Kim G.Larsen, Robin Milner, A Complete Protocol Verification using Relativized Bisimulation, LFCS Report ECS-LFCS-86-13, Department of Computer Science, University of Edinburgh, September 1986

[Lichtenstein85] O. Lichtenstein, A. Pnueli, Checking that Finite State Concurrent Programs Satisfy Their Linear Specifications, Proc. 12th ACM Symp. on Principles of Programming Languages, New Orleans, 1985, pp 97-107.

[Lin87] F.J.Lin, P.M.Chu, M.T.Liu, Protocol Verification Using Reachability Analysis, The State Space Explosion Problem and Relief Strategies, ACM SIGCOMM, Stowe, Vermont 1987, Computer Communication Review, vol.17, #5, 1987

[Manna74] Z. Manna, A.Pnueli, An Axiomatic Approach to Total Correctness of Programs, Acta Informatica, 3:243-263, 1974

[Manna81] Z. Manna, A.Pnueli, The Temporal Framework for Concurrent Program, in The Correctness Problem in Computer Science, R.S. Boyer, J.S. Moore, eds., Academic Press (1981) pp.215-274

[Masapati87] Algorithms for the Reduction of Timed Finite State Graphs, ACM SIGCOMM 1987, Computer Communication Review, vol.17, #5, 1987

[Milner80] R.Milner, A Calculus of Communicating Systems, Springer LNCS 92, 1980

[Milner83] R.Milner, Calculi for Synchrony and Asynchrony, Theor. Comp. Sci. 25:3 (1983) pp 267-311

[Neiger87] Gil Neiger, Sam Toueg, Substituting for Real Time and Common Knowledge in Asynchronous Distributed Systems, Proc. 6th Annual ACM Symp. on Principles of Distributed Systems, Vancouver, August 1987.

[Nguyen86] Van Nguyen, Alan Demers, David Gries, Susan Owicki, A Model and Temporal Proof System for Networks of Processes, Distributed Computing (1986)1:7-25, Springer Verlag, 1986

[Owicki76] Susan S. Owicki, David Gries, Verifying Properties of Parallel Programs, CACM, 19(5):279-285, May 1976.

[Owicki82] Susan S. Owicki, Leslie Lamport, Proving Liveness Properties of Concurrent Programs, ACM TOPLAS, Vol.4, No.3, July 1982, pp455-495.

[Pachl87] Jan Pachl, Protocol Description and Analysis Based on a State Transition Model with Channel Expressions, IFIP Int. Conf. on Protocol Specification, Testing and Verification VI, Montreal 1986, North Holland 1987

[Parrow86] Joachim Parrow, Fairness Properties in Process Algebra, PhD thesis, Uppsala University, January 1986

[Parrow88] Joachim Parrow, Verifying a CSMA/CD-protocol with CCS, IFIP Int. Conf. on Protocol Specification, Testing and Verification VIII, Atlantic City 1988

[Reed88] Joylyn Reed, Raymond Yeh, Specification and Verification of Liveness Properties of Cyclic, Concurrent Properties, ACM TOPLAS, vol.10, #1, January 1988

[Richier87] J.L.Richier, C.Rodrigues, J.Sifakis, J.Voiron, Verification in Xesar of the Sliding Window Protocol, IFIP Int. Conf. on Protocol Specification, Testing and Verification VII, Zürich 1987, North Holland 1988

[Rudin83] Harry Rudin, From Formal Protocol Specification Towards Automated Performance Prediction, IFIP Int. Conf. on Protocol Specification, Testing and Verification III, Zürich 1983, North Holland 1983

[Shankar83] A.Udaya Shankar, Simon S. Lam, An HDLC Protocol Specification and Its Verification Using Imge Protocols, ACM TOCS vol.1,#4, November 1983

[Schwartz82] R.L. Schwartz, M. Melliar-Smith, From State Machines to Temporal Logic: Specification Methods for Protocol Standards, IEEE Trans. Comm, December 1982

[Vardi86] Moshe Y. Vardi, Pierre Wolper, An Automata-Theoretic Approach to Automatic Program Verification, Proc. IEEE Symp. on Logic in Computer Science, Cambridge, June 1986

[Walker87] Introduction to a Calculus of Communicating Systems, Laboratory for Foundations of Computer Science, Department of Computer Science Edinburgh, February 1987.

[West78] C.H.West and P. Zafiropulo, Automated Validation of a Communication Protocol: The CCITT X.21 Recommendation, IBM Journal of Research and Development, Vol.2, January 1978 pp 60-71.

[Zafiropulo80] Pietro Zafiropulo, Colin H. West, Harry Rudin, D.D.Cowan, D.Brand, Towards Analyzing and Synthesizing Protocols, IEEE Trans Comm., Vol.COM-28, No.4, April 1980

[Zic87] John J. Zic, Extension to Communicating Sequential Processes to Allow Protocol Performance Specification, ACM SIGCOMM 1987, Computer Communication Review, vol.17, #5, 1987